M&G No. 60001.0283US01/304803.01

REPRESENTING NON-STRUCTURED FEATURES IN A WELL FORMED DOCUMENT

Related Applications

This patent application is a continuation-in-part application under 35 United States Code § 120 of United States Patent Application No. 10/187,060 filed on June 28, 2002, which is incorporated herein by reference. An exemplary schema in accordance with the present invention is disclosed beginning on page 11 in an application entitled "Mixed Content Flexibility," Serial No. _____, Docket No. 60001.0275US01, filed December 2, 2003, which is hereby incorporated by reference in its entirety.

Background of the Invention

Markup Languages have attained wide popularity in recent years. One type of markup language, Extensible Markup Language (XML), is a universal language that provides a way to identify, exchange, and process various kinds of data. For example, XML is used to create documents that can be utilized by a variety of application programs. Elements of an XML file have an associated namespace and schema.

In XML, a namespace is a unique identifier for a collection of names that are used in XML documents as element types and attribute names. The name of a namespace is commonly used to uniquely identify each class of XML document. The unique namespaces differentiate markup elements that come from different sources and happen to have the same name.

XML Schemata provide a way to describe and validate data in an XML environment. A schema states what elements and attributes are used to describe content in an XML document, where each element is allowed, what types of text contents are allowed within it and which elements can appear within which other elements. The use of schemata ensures that the document is structured in a consistent manner. Schemata

may be created by a user and generally supported by an associated markup language, such as XML. By using an XML editor, the user can manipulate the XML file and generate XML documents that adhere to the schema the user has created. XML documents may be created to adhere to one or more schemata.

The XML standard is by many considered the ASCII format of the future, due to its expected pervasiveness throughout the hi-tech industry in the coming years. Recently, some word-processors have begun producing documents that are somewhat XML compatible. For example, some documents may be parsed using an application that understands XML.

In XML, it is necessary to maintain a well formed document. Generally, this means that tags within the XML document do not overlap. There are a number of features in word processors, however, that are allowed to span arbitrary ranges. These features include features such as comments, bookmarks, document protection, and the like. What is needed is a way to represent these features in XML.

Summary of the Invention

The present invention is directed towards respresenting non-structured features that are common with word-processors such that these elements can be recognized and parsed separately from other elements within an XML document.

According to one aspect of the invention, non-structured features are represented as well formed in XML. Some of the features that may span arbitrary ranges include features such as comments, bookmarks, document protection, and the like.

According to another aspect of the invention, empty tags are used to mark the start and end of a feature that may span other features. These elements can be recognized and parsed separately from other elements.

According to yet another aspect of the invention, the word-processing documents may be parsed by applications that understand XML. The XML word-processing documents may be manipulated on a server, or anywhere even when the word-processor creating the XML document is not present.

Brief Description of the Drawings

FIGURE 1 illustrates an exemplary computing device that may be used in one exemplary embodiment of the present invention;

FIGURE 2 is a block diagram illustrating an exemplary environment for practicing the present invention;

FIGURE 3 illustrates an exemplary ML files including some formatted text;

FIGURE 4 illustrates an exemplary ML file with a bookmark spanning two paragraphs; and

FIGURE 5 illustrates a process for representing non-structured features in a well formed document, in accordance with aspects of the invention.

Detailed Description of the Preferred Embodiment

Throughout the specification and claims, the following terms take the meanings explicitly associated herein, unless the context clearly dictates otherwise.

The terms "markup language" or "ML" refer to a language for special codes within a document that specify how parts of the document are to be interpreted by an application. In a word-processor file, the markup language specifies how the text is to be formatted or laid out, whereas in a particular customer schema, the ML tends to specify the text's meaning according to that customer's wishes (e.g., customerName, address, etc.) The ML is typically supported by a word-processor and may adhere to the rules of other markup languages, such as XML, while creating further rules of its own.

The term "element" refers to the basic unit of an ML document. The element may contain attributes, other elements, text, and other building blocks for an ML document.

The term "tag" refers to a command inserted in a document that delineates elements within an ML document. Each element can have no more than two tags: the start tag and the end tag. It is possible to have an empty element (with no content) in which case one tag is allowed.

The content between the tags is considered the element's "children" (or descendants). Hence other elements embedded in the element's content are called "child elements" or "child nodes" or the element. Text embedded directly in the content of the element is considered the element's "child text nodes". Together, the child elements and the text within an element constitute that element's "content".

The term "attribute" refers to an additional property set to a particular value and associated with the element. Elements may have an arbitrary number of attribute settings associated with them, including none. Attributes are used to associate additional information with an element that will not contain additional elements, or be treated as a text node.

Illustrative Operating Environment

With reference to FIGURE 1, one exemplary system for implementing the invention includes a computing device, such as computing device 100. In a very basic configuration, computing device 100 typically includes at least one processing unit 102 and system memory 104. Depending on the exact configuration and type of computing device, system memory 104 may be volatile (such as RAM), non-volatile (such as ROM, flash memory, etc.) or some combination of the two. System memory 104 typically includes an operating system 105, one or more applications 106, and may include program data 107. In one embodiment, application 106 may include a word-processor application 120 that further includes non-structured features 122. This basic configuration is illustrated in FIGURE 1 by those components within dashed line 108.

Computing device 100 may have additional features or functionality. For example, computing device 100 may also include additional data storage devices (removable and/or non-removable) such as, for example, magnetic disks, optical disks, or tape. Such additional storage is illustrated in FIGURE 1 by removable storage 109 and non-removable storage 110. Computer storage media may include volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information, such as computer readable instructions, data structures, program modules, or other data. System memory 104, removable storage 109 and non-removable storage 110 are all examples of computer storage

media. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by computing device 100. Any such computer storage media may be part of device 100. Computing device 100 may also have input device(s) 112 such as keyboard, mouse, pen, voice input device, touch input device, etc. Output device(s) 114 such as a display, speakers, printer, etc. may also be included. These devices are well know in the art and need not be discussed at length here.

Computing device 100 may also contain communication connections 116 that allow the device to communicate with other computing devices 118, such as over a network. Communication connection 116 is one example of communication media. Communication media may typically be embodied by computer readable instructions, data structures, program modules, or other data in a modulated data signal, such as a carrier wave or other transport mechanism, and includes any information delivery media. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, RF, infrared and other wireless media. The term computer readable media as used herein includes both storage media and communication media.

Representing Non-Structured Features in a Well Formed Document

Generally, the present invention is directed at representing non-structured features common with word-processors such that these elements can be recognized and parsed separately from other elements

In XML, it is necessary to maintain a well formed document. Generally, this means that tags within the XML document do not overlap. There are a number of features in word processors, however, that are allowed to span arbitrary ranges. These

features include features such as comments, bookmarks, document protection, and the like.

FIGURE 2 is a block diagram illustrating an exemplary environment for practicing the present invention. The exemplary environment shown in FIGURE 2 is a word-processor environment 200 that includes word-processor 120, ML file 210, ML Schema 215, and ML validation engine 225.

In one embodiment, word-processor 120 has its own namespace or namespaces and a schema, or a set of schemas, that is defined for use with documents associated with word-processor 120. The set of tags and attributes defined by the schema for word-processor 120 define the format of a document to such an extent that it is referred to as its own native ML. Word-processor 120 internally validates ML file 210. When validated, the ML elements are examined as to whether they conform to the ML schema 215. A schema states what tags and attributes are used to describe content in an ML document, where each tag is allowed, and which tags can appear within other tags, ensuring that the documentation is structured the same way. Accordingly, ML 210 is valid when structured as set forth in arbitrary ML schema 215.

ML validation engine 225 operates similarly to other available validation engines for ML documents. ML validation engine 225 evaluates ML that is in the format of the ML validation engine 225. For example, XML elements are forwarded to an XML validation engine. In one embodiment, a greater number of validation engines may be associated with word-processor 120 for validating a greater number of ML formats.

FIGURE 3 illustrates an exemplary ML file including some formatted text, in accordance with aspects of the present invention. ML file 300 includes ML elements. An element in a markup language usually includes an opening tag (indicated by a "<" and ">"), some content, and a closing tag (indicated by a "</" and ">").

There are enough ML elements for an application that understands XML to fully recreate the document from a single XML file. Hint tags may also be included that provide information to an application to help understand the content of the file.

There are a number of fundamental rules when using XML. One of these rules is called "well-formed ness." This means that the XML markup must not overlap. Here is an example of XML that is **not** well formed:

```
<root>
    <title>
        Here is my title
        <subTitle>
            Here is my sub title
    </title>
        </subTitle>
</root>
```

Note how the <subTitle> tag starts inside of the <title> tag, but the <subTitle> ends outside of the <title> tag. In order for this document to be a well formed XML document, it should look like the following:

```
<root>
    <title>
        Here is my title
        <subTitle>
            Here is my sub title
        </subTitle>
    </title>
</root>
```

There are a number of word-processing features that consist of some type of "structure" applied to a range of text. A bookmark for instance can be applied to a selection of text. For purposes of this disclosure, assume that a word-processing

bookmark is identified by the <w:bookMark> tag. An example will be presented to illustrate.

The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog.

If one were to apply a bookmark called "2nd sentence" to the 2nd sentence of the above paragraph, the XML representation of that might look something like what is illustrated in FIGURE 3.

While this style of tags works for representing the paragraph with the <w:p> tag, and the bookmark with the <w:bookMark> tag. This approach does not always work. What if there were two paragraphs with a bookmark spanning the paragraphs.

FIGURE 4 illustrates an exemplary ML file with a bookmark spanning two paragraphs. The text shown in FIGURE 4 is:

Here is my first paragraph
Here is my second paragraph

Using the approach as illustrated in FIGURE 4, if you were to apply a bookmark from "first" within the first paragraph to "second" within the second paragraph, the XML would look something like:.

<w:p>
Here is my
<w:bookMark name="2nd sentence">
First paragraph
</w:p>
<w:p>

Here is my

</w:bookMark>

second paragraph

</w:p>

The above example is not well formed, as the bookmark tag overlaps the paragraph tags. To create a well formed XML representation, two tags will be used for for objects like bookmarks. According to one embodiment, there will be a <w:bookmarkStart> tag and <w:bookmarkEnd> tag to represent bookmarks. So the above example would be represented as shown in FIGURE 4.

Since the <w:bookmarkStart> and <w:bookmarkEnd> tags are both empty tags, they don't have the problem of wrapping the <w:p> tag. Instead, they are two empty tags that are associated with each other by the id attribute contained within both of the tags. With this method, it is possible to show where a bookmark starts and ends, while still maintaining a well formed document.

According to one embodiment, using empty tags is used for a number of different features in the word-processor, including: Range Level permissions; Bookmarks; Comments; Tracked changes; Spelling Errors; and Grammar Errors.

Bookmarks are used in Word processing documents for a variety of reasons. Bookmarks allow a user to call attention to part of a document without actually altering the document. A bookmark allows a user to easily get back to that point in the document.

Bookmarks become even more powerful with XML. Since XML is a text based format that is easily readable and parseable, bookmarks become a great way to getting into a specific portion of a rich document. Bookmarks may also be used within XML documents to index the documents based on their bookmarks. For example, documents stored on a server may be bookmarked and then indexed.

Bookmarks not only identify a key areas in a document, they also allow a user to select a range within a document. In other words, this is analogous to taking a

book, and instead of just inserting a bookmark on a single page, a user could highlight a specific section of text within the book.

Since bookmarks can be applied to a range, one could use an XML parser to show the textual values of all bookmarks in a specific document or in a group of documents.

This provides bookmarks something that is not possible using most XML schemas. Just as an example, take the following example schema for a memo. Assume the following elements: "To:", "From:", "Subject:" and "Message:".

These elements would allow a user to create a structured memo that could easily be routed to the proper recipient. An example memo is as follows:

<memo><from>Brian</from>
<to>Scott</to>
<subject>Hello</subject>
<message>Hey there, how's it going. Did you finish the task of mailing the feedback?</message>
</memo>

Now, what if the actual task of "mailing the feedback" were of interest to certain people? It's obvious that tasks aren't always going to be in a memo, so there probably wouldn't be an actual "task" element in the memo schema. Even if there were, there are probably a number of other types of things that can randomly appear in a memo that some people may want to flag, but that wouldn't appear in the memo schema.

With the bookmark feature, it is possible to flag that bit of text within the "message" element, so that XML parsers could easily parse through not just the memo, but parse through relevant bits of data within the actual message. The following is an exemplary way to bookmark the "mailing the feedback" section.

<memo><from>Brian</from>
<to>Scott</to>

```
<subject>Hello</subject>
<message>Hey there, how's it going. Did you finish the task of
<w:bookmarkStart name="Feedback" id="bk2"/>
mailing the feedback?
<w:bookmarkEnd id ="bk2"/>
</message>
</memo>
```

FIGURE 5 illustrates a process for representing non-structured features in a well formed document, in accordance with aspects of the invention. After a start block, process 500 flows to block 510 where the start tag is determined. The start tag may be located anywhere within the document. Moving to block 515 the end tag location is determined. As discussed above, the end tag may span other elements while still maintaining a well formed document. Flowing to block 520, the start and end tags are created. According to one embodiment of the invention, the tags reference each other through the use of an "id" that is common to both tags. Moving to block 525, the tags are placed within the document. Flowing to block 530, the XML file that is generated is well formed even though there are non-structured elements contained within the file. The process then moves to an end block and returns to processing other actions.

Here is an exemplary definition of a paragraph, in accordance with aspects of the invention. Some of the elements, include: **aml:annotation; proofErr; permStart; permEnd**

```
<xsd:complexType name="pElt">
        <xsd:sequence>
            <xsd:element name="pPr" type="pPrElt" minOccurs="0">
                <xsd:annotation>
```

```
                    <xsd:documentation>Paragraph
properties</xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:choice minOccurs="0" maxOccurs="unbounded">
                <xsd:element ref="aml:annotation" minOccurs="0"
maxOccurs="unbounded"></xsd:element>
                <xsd:element name="proofErr" type="proofErrElt">
                    <xsd:annotation>
                        <xsd:documentation>Proofing state
(grammar, spelling, etc)</xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
                <xsd:element name="permStart" type="permStartElt">
                    <xsd:annotation>
                        <xsd:documentation>Range Protection
Permission Start</xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
                <xsd:element name="permEnd" type="permElt">
                    <xsd:annotation>
                        <xsd:documentation>Range Protection
Permission End</xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
                <xsd:element name="r" type="rElt">
                    <xsd:annotation>
                        <xsd:documentation>Run element. This is
the leaf container for data in a Word document -- text, pictures,
etc</xsd:documentation>
```

```
                    </xsd:annotation>
                </xsd:element>
                <xsd:element name="fldSimple" type="simpleFieldType"
minOccurs="1" maxOccurs="unbounded">
                    <xsd:annotation>
                        <xsd:documentation>Simple word field
(with plain text instructions). These are run-time calculated entities in word (eg: page
numbers, etc)</xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
                <xsd:element name="hlink" type="hLinkType">
                    <xsd:annotation>
                        <xsd:documentation>hyperlink element
(analagous to HTML &lt;a href=...&gt; tag)</xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
                <xsd:element name="subDoc" type="subDocElt">
                    <xsd:annotation>
                        <xsd:documentation>Link to sub
document (i.e. master document / sub documents)</xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
                <xsd:element name="ruby" type="rubyElt">
                    <xsd:annotation>
                        <xsd:documentation>asian layout:
phonetic guide (ruby text)</xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
            </xsd:choice>
        </xsd:sequence>
```

```
</xsd:complexType>
```

According to one embodiment, the following is a definition the **aml:annotation** element:

```
<xsd:element name="annotation" type="aml:AnnType">
        <xsd:unique name="uniqueContentAnchorIdsInsideAnnotations">
            <xsd:selector xpath="aml:content"></xsd:selector>
            <xsd:field xpath="@id"></xsd:field>
        </xsd:unique>
        <xsd:unique name="uniqueContextAnchorIdsInsideAnnotations">
            <xsd:selector xpath="aml:context"></xsd:selector>
            <xsd:field xpath="@id"></xsd:field>
        </xsd:unique>
    </xsd:element>
    <xsd:complexType name="AnnType" mixed="false">
        <xsd:sequence>
            <xsd:element ref="aml:arc" minOccurs="0"
maxOccurs="1"></xsd:element>
            <xsd:element ref="aml:context" minOccurs="0"
maxOccurs="unbounded"></xsd:element>
            <xsd:element ref="aml:content" minOccurs="0"
maxOccurs="unbounded"></xsd:element>
            <xsd:element ref="aml:property" minOccurs="0"
maxOccurs="unbounded"></xsd:element>
            <xsd:any namespace="##other" processContents="lax"
minOccurs="0" maxOccurs="unbounded"></xsd:any>
        </xsd:sequence>
```

```
                <xsd:attribute name="type" type="type"
fixed="extended"></xsd:attribute>
                <xsd:attribute name="id" type="aml:idType"
use="required"></xsd:attribute>
                <xsd:attribute name="author" type="aml:authorType"
use="optional"></xsd:attribute>
                <xsd:attribute name="createdate" type="aml:dateType"
use="optional"></xsd:attribute>
                <xsd:anyAttribute namespace="##other"
processContents="lax"></xsd:anyAttribute>
        </xsd:complexType>
```

Where the attribute type xsd:anyAttribute, and element type xsd:any are referenced, is where the word processor specific information can go. Those attributes are described in the attribute group "wordAnnotationGroup":

```
<xsd:attributeGroup name="wordAnnotationGroup">
                <xsd:attribute name="type" type="annotationValuesType"
use="required">
                        <xsd:annotation>
                                <xsd:documentation>The Word element expressed by this
AML Annotation Tag.</xsd:documentation>
                        </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="name" type="stringType" use="optional">
                        <xsd:annotation>
                                <xsd:documentation>For bookmarks, specifies the
bookmark name.</xsd:documentation>
                        </xsd:annotation>
                </xsd:attribute>
```

```
<xsd:attribute name="initials" type="stringType" use="optional">
    <xsd:annotation>
        <xsd:documentation>For bookmarks denoting the range of a comment, specifies the initials of the comment author.</xsd:documentation>
    </xsd:annotation>
</xsd:attribute>
<xsd:attribute name="col-first" type="decimalNumberType" use="optional">
    <xsd:annotation>
        <xsd:documentation>For table bookmarks, specifies the column this bookmark begins in.</xsd:documentation>
    </xsd:annotation>
</xsd:attribute>
<xsd:attribute name="col-last" type="decimalNumberType" use="optional">
    <xsd:annotation>
        <xsd:documentation>For table bookmarks, specifies the column this bookmark ends in.</xsd:documentation>
    </xsd:annotation>
</xsd:attribute>
<xsd:attribute name="original" type="stringType" use="optional">
    <xsd:annotation>
        <xsd:documentation>The original numbering on display field rev marking.</xsd:documentation>
    </xsd:annotation>
</xsd:attribute>
<xsd:attribute name="displacedBySDT" type="displacedBySDTValue" use="optional">
    <xsd:annotation>
```

<xsd:documentation>When bookmarks border SDTs (Structured Document Tags), use this attribute to help ensure that they are inserted into the document next to the SDTs. When we displace the SDTs, we also displace the bookmarks next to them so the intended result is in the XML file.</xsd:documentation>
</xsd:annotation>
</xsd:attribute>
</xsd:attributeGroup>

The values used in the "type" attribute are described below:

```
<xsd:simpleType name="annotationValuesType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Word.Insertion"></xsd:enumeration>
        <xsd:enumeration value="Word.Deletion"></xsd:enumeration>
        <xsd:enumeration value="Word.Formatting"></xsd:enumeration>
        <xsd:enumeration value="Word.Bookmark.Start"></xsd:enumeration>
        <xsd:enumeration value="Word.Bookmark.End"></xsd:enumeration>
        <xsd:enumeration value="Word.Comment.Start"></xsd:enumeration>
        <xsd:enumeration value="Word.Comment.End"></xsd:enumeration>
        <xsd:enumeration value="Word.Insertion.Start"></xsd:enumeration>
        <xsd:enumeration value="Word.Insertion.End"></xsd:enumeration>
        <xsd:enumeration value="Word.Deletion.Start"></xsd:enumeration>
```

```
                    <xsd:enumeration
value="Word.Deletion.End"></xsd:enumeration>
                    <xsd:enumeration value="Word.Comment"></xsd:enumeration>
                    <xsd:enumeration
value="Word.Numbering"></xsd:enumeration>
                </xsd:restriction>
            </xsd:simpleType>
```

So, in the case of a Bookmark, the beginning tag would look something like:

```
<aml:annotation aml:id="0" w:type="Word.Bookmark.Start" w:name="myBookmark"
/>
```

And the end of the bookmark would look something like:

```
<aml:annotation aml:id="0" w:type="Word.Bookmark.End" />
```

According to one embodiment, the following is a list of exemplary **proofErr** types:

```
<xsd:simpleType name="proofErrType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="spellStart"></xsd:enumeration>
            <xsd:enumeration value="spellEnd">
                <xsd:annotation>
                    <xsd:documentation>We take the negative value
of ptl and then subtract one to get the end constant.</xsd:documentation>
                </xsd:annotation>
            </xsd:enumeration>
            <xsd:enumeration value="gramStart"></xsd:enumeration>
            <xsd:enumeration value="gramEnd">
                <xsd:annotation>
                    <xsd:documentation>We take the negative value
of ptl and then subtract one to get the end constant.</xsd:documentation>
```

```
                    </xsd:annotation>
                </xsd:enumeration>
            </xsd:restriction>
        </xsd:simpleType>
```

According to one embodiment, the following is a definition of the **permStart** element:

```
<xsd:complexType name="permStartElt">
        <xsd:complexContent>
            <xsd:extension base="permElt">
                <xsd:attribute name="edGrp" type="edGrpType"
use="optional">
                    <xsd:annotation>
                        <xsd:documentation>Group with edit
permissions</xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="ed" type="stringType"
use="optional">
                    <xsd:annotation>
                        <xsd:documentation>User with edit
permissions</xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="col-first"
type="decimalNumberType" use="optional"></xsd:attribute>
                <xsd:attribute name="col-last"
type="decimalNumberType" use="optional"></xsd:attribute>
            </xsd:extension>
        </xsd:complexContent>
```

```
</xsd:complexType>
```

According to one embodiment, the following is a definition for the **permEnd** element:

```
<xsd:complexType name="permElt">
    <xsd:attribute name="id" type="stringType" use="required">
        <xsd:annotation>
            <xsd:documentation>Id for this
Permission</xsd:documentation>
        </xsd:annotation>
    </xsd:attribute>
    <xsd:attribute name="displacedBySDT" type="displacedBySDTValue"
use="optional">
        <xsd:annotation>
            <xsd:documentation>When bookmarks border SDTs
(Structured Document Tags), use this attribute to ensure that they are inserted into the
document next to the SDTs. We use this attribute because SDTs appear in our XML,
how they logically appear in the Word document, but not neccessarily in the same
location as they are in the document. When we displace the SDTs, we also displace the
bookmarks next to them so the intended result is in the XML file.</xsd:documentation>
        </xsd:annotation>
    </xsd:attribute>
</xsd:complexType>
```

The above specification, examples and data provide a complete description of the manufacture and use of the composition of the invention. Since many embodiments of the invention can be made without departing from the spirit and scope of the invention, the invention resides in the claims hereinafter appended.